WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, December 01, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L21	l19 not L20	7
☐	L20	yoaz.in.	46
☐	L19	l17 and L18	11
☐	L18	vot$ or majorit$	310038
☐	L17	l12 and L16	51
☐	L16	(predict$) NEAR4 (bank or banks)	373
☐	L15	l8 and l13	6
☐	L14	l9 and L13	1
☐	L13	l4 and L12	858
☐	L12	(cach$ or bank$).ab,ti. and cach$ and (bank or banks)	2897
☐	L11	l5 and l8	1
☐	L10	l5 and L9	0
☐	L9	(select$ or choos$ or pick) SAME (vot$)	9126
☐	L8	vot$	82325
☐	L7	l5 and L6	83
☐	L6	((select$ or choos$ or pick) NEAR4 (bank or banks)).ab,ti.	1971
☐	L5	l3 and L4	211
☐	L4	(select$ or choos$ or pick) NEAR4 (bank or banks)	12558
☐	L3	l1 and L2	747
☐	L2	banks.ab,ti.	63872
☐	L1	(cache or caching).ab,ti.	32928

END OF SEARCH HISTORY

Detailed Description Text (33):
The present invention, an address generator of which is depicted in FIG. 3, also reflects the recognition that the contents of the base register have a very high probability of having a 0 value for certain bits, e.g., 58-63, of the base register. For example, for commercial reduced instruction set computing code, the majority of the most commonly referenced objects, i.e., chunks of memory, are highly aligned, usually along 32-byte or larger boundaries. A highly aligned object starts on a definite boundary such as the beginning of a cache page or cache line. The compiler can enhance the situation by forcing the register to be aligned by adding a given value to the address in the base register, for which subsequent compensation takes the form of subtracting the given value from subsequent values in the displacement register.

Detailed Description Text (34):
The address generator of FIG. 3 generates displacement-type, or D-form, addresses based upon the assumption that bits 58-63 of the base register will be zero. This makes it possible for the compiler to limit to no more than two the number of same cycle random loads to the same bank. Without having to deal with more than two random loads per cycle per bank, the cycle memory can be formed from only two logical parts and yet be functionally equivalent to a multi-ported memory.

Detailed Description Text (35):
By limiting the processor associated with the cache memory of the present invention to using only base-plus-displacement-type addressing, and by recognizing that the bits 58-63 of the base register will almost always be 0, the compiler can determine the address of any storage instruction to the cache memory with a very high degree of certainty at compile time. In other words, the bits 10-12 of the displacement register become a very accurate predictor of the bank within the cache memory that will be the subject of a storage instruction.

Detailed Description Text (36):
The adder 306 of FIG. 3 operates upon bits 58-63 of the base register 302, which have a very high probability of always being 0, and upon bits 10-12 of the displacement register 304. For effective address generation purposes, only bits 58-60 of the 64-bit sum produced by the adder 306 are needed. As such, the 3-8 decoder of FIG. 3 operates only upon bits 58-60 of the sum output by the adder 306. The eight output lines of the decoder 308 are used as the bank select lines BS0 to BS7.

Detailed Description Text (18):
Turning to FIGS. 3A to 3C, further details of the CMC 210 and its connections to the L2 cache 208 are shown. Turning to FIG. 3A, the L2 cache 208 includes two 32-bit wide synchronous burst SRAMs 300 and 302. These are preferably MT58LCK36B2 32K by 36 synchronous burst SRAMs by Micron Semiconductor, Inc. Two burst SRAMs 300 and 302 are used in the disclosed embodiment because the processor data bus is 64 bits wide. The burst SRAM 300 is connected to the processor 200 data bus lines PD[63:48, 31:16], while the burst SRAM 302 is connected to lines PD[47:32, 15:0]. Thus, each of the burst SRAMs 300 and 302 provides half of the data to and from the processor 200 data bus. The processor 200 address bus PA, however, is identically connected to each of the burst SRAMs 300 and 302. PA[16:3] is provided to fourteen address inputs of each of the burst SRAMs 300 and 302, thus providing a 14-bit index. The fifteenth address input A[14] of the burst SRAMs 300 and 302, however, is provided by a cache memory way selection signal CMWAY, which according to the invention, provides for a two-way cache using the single bank of cache data memory. This signal is generated by the CMC 210, and will be discussed below in conjunction with FIGS. 4A-4C. Other inputs provided by the CMC 210 to the burst SRAMs 300 and 302 are an active low output enable signal COE0* that is provided to the chip output enable inputs, a cache address strobe signal CADS0* that is provided to the controller address strobe input of the burst SRAMs 300 and 302, a cache chip select signal CCS0* that is provided to the chip enable input of the burst SRAMs 300 and 302, a cache advance strobe CADV* that is provided to the cache advance input of the burst SRAMs 300 and 302, and cache write enable signals CWE*[7:0], the appropriate ones of which are provided to each of the burst SRAMs 300 and 302 corresponding to the data lines PD to which the burst SRAMs 300 and 302 are connected. These signals are further described below in conjunction with FIGS. 4A-C, 6A-K, and 25A-F.

Detailed Description Text (20):
In prior art systems, if the burst SRAMs 300 and 302 were used to implement a two-way set-associative cache, four such chips would be necessary. FIG. 3B illustrates such a system. This is an alternative embodiment for implementing one aspect according to the invention. FIG. 3B illustrates a first bank 310 of such a two-way set-associative cache. In such a two-bank mode, all 15 bits of the first bank 310 address inputs are connected to PA[17:3], with the CMC 210 providing COE0* as before to the first bank 310, but now providing a second enable signal COE1* as the chip output enable to the second way, formed by a second bank 320. Similarly, a second chip select signal CCS1* is provided to the second bank 320, and now each bank requires two burst SRAMs 312 and 314, and 322 and 324, to handle the 64-bit data path of PD[63:0]. Even in such a prior art system, advantages can result when the last code read or last code plus data read way prediction according to the invention is used. This is further discussed below, especially in conjunction with FIG. 33. Selection in FIG. 3B between the two ways formed by banks 310 and 320 requires either waiting until tag RAMS 364 or 366 in the CMC 210 have returned a hit to one way or the other way and then selecting the appropriate way of the banks 310 or 320 using COE0* or COE1*, or alternatively, predicting which way of the banks 310 or 320 the next processor 200 operation would be directed to, typically based on a least recently used algorithm from data in the tag RAMs 364 or 366. In either case, however, a separate physical bank of chips would be necessary for each way.

Detailed Description Text (21):

L21: Entry 5 of 7 File: USPT May 26, 1998

DOCUMENT-IDENTIFIER: US 5758142 A
TITLE: Trainable apparatus for predicting instruction outcomes in pipelined processors

Abstract Text (1):
A predictor which chooses between two or more predictors is described. The predictor includes a first component predictor which operates according to a first algorithm to produce a prediction of an action and a second component predictor which operates according to a second algorithm to produce a prediction of said action. The predictor also includes means, coupled to each of said first and second predictors, for choosing between predictions provided from said predictors to provide a prediction of the action from the predictor. The predictor can be used to predict outcomes of branches, cache hits, prefetched instruction sequences, and so forth.

Brief Summary Text (3):
As it is known in the art, computer systems have become ubiquitous. In particular, one type of computer system widely employed includes a so called pipelined processor. In a pipelined processor, instructions are decomposed into assembly-like stages and the instruction is operated on at each of the stages for each clock cycle in the pipeline. Illustratively, a pipelined stage includes an instruction fetch stage in which instructions are fetched in one or several cycles from a cache memory, an instruction decode stage in which an instruction's op-code, (i.e., a portion of the instruction which determines the function of the instruction) is examined to ascertain the function of the instruction and, thus, the resources needed by the instruction, as well as instruction issue stage and so forth.

Brief Summary Text (15):
Other dependencies such as data dependencies are encountered in pipelined processors. These data dependencies also require the processor to wait or flush the pipeline. One type of data dependency which occurs is a cache dependency, more commonly referred to as a cache miss.

Brief Summary Text (16):
One feature of modern processors is the use of very, high speed cache memory on or close to a semiconductor chip which embodies the central processing unit of the processor. So-called static random access memory, is used to provide a small but very fast and accessible cache for storage of data which the processor is currently or is expected to presently use. Generally, the period of time or latency from which an instruction requests data from a cache is substantially less than the time that it takes to retrieve the same data from main memory. That is, when a processor executes a load instruction, i.e., retrieves data from a cache or main memory to load a register in the processor, the processor issues an address and the address is examined by the cache. The cache generally examines a portion of the address bits against tags stored in a tag store associated with the cache to determine whether the requested data is resident in the cache. There are two basic types of caches, direct map caches and associative caches. With either type of cache, if the data requested by the processor is in the cache, a so-called cache hit is provided indicating that the cache can quickly supply the data to the processor. However, if the data is not in the cache, the data must be retrieved from either one or more

lower level (and hence slower) caches or from main memory. Thus, with a caches miss generally, the processor stalls or delays processing instructions until additional time has elapsed for the data to be retrieved from the location containing the requested data until it can be provided to the processor.

Brief Summary Text (17):
The problem associated with caches in pipelined processors employing instruction schedulers is that in the event that an instruction scheduler schedules an instruction, for example a load instruction to issue from the cache, it would be desirable to know whether or not the data will be in the cache prior to the instruction actually being executed. Therefore, a prediction technique which can predict cache hits and cache misses would be desirable. In the event a prediction technique is provided, the instruction scheduler can use this information to hold off scheduling instructions that need the load result operand, while scheduling other instructions which are indepentant of the load operand.

Drawing Description Text (8):
FIG. 6 is a block diagram of a pipelined processor employing a cache hit predictor in accordance with a still further aspect of the present invention;

Drawing Description Text (9):
FIG. 7 is a block diagram of a cache hit predictor for use in the processor shown in FIG. 6;

Detailed Description Text (10):
Therefore, the choosing predictor 10 includes a choosing circuit 20 comprised of the aforementioned multiplexer 24, as well as control logic 25, which is used to update or train a choosing table 22. In the embodiment shown in FIG. 1, the choosing table 22 is a choosing counter table and is addressed via the output of register 11a, here 13 bits of the program counter (PC) 11 for the particular decoded branch type of instruction, as previously mentioned. Alternatively, the choosing table 22 could be a register stack whose states are used to determine majority direction. In the present embodiment, the PC serves to access the counter table 22 here comprised of two bit counters 22.sub.a -22.sub.k-1. These counters provide their MSB as the selection bit to determine whether the multiplexer 24 will provide at the output 24a thereof a branch prediction from the predictor 12 or the predictor 16. The counter 22 is updated via training logic 25 which has as inputs, the prediction outputs of each component predictor 12, 16, as well as the branch resolution signal 26 provided from a branch resolution circuit not shown but which can be the type as described in the above mentioned co-pending application.

Detailed Description Text (20):
This dual stage global predictor scheme can work better than the single stage scheme 16, described in conjunction with FIGS. 1 and 2 since using history bits allows patterns of behavior to be recognized at a particular global history compared to just majority direction as is provided from just using counters.

Detailed Description Text (23):
The instructions are fed to a plurality of execution boxes (EBOX.sub.1 -EBOX.sub.i) here 34a-34i, as shown via bus 31. Each of the execution boxes EBOX.sub.1 - EBOX.sub.i includes a general purpose register file 35a arithmetic logic unit 35b and first level cache 35c and is generally of the type also described in the above-identified patent application although other types can alternatively be used. The processor 20 further includes a cache box 36 including a tag store 37a, a write buffer 37b, as well as second and third level caches 37c and 37d, and is generally of a type also described in conjunction with the above-mentioned co-pending patent application.

Detailed Description Text (24):
The processor 20 further includes a cache hit prediction choosing predictor 50, as

will be generally described in conjunction with FIG. 7. The cache hit predictor 50 is used to generate a prediction for load type of instructions as to whether data required by the load instruction is or will be available in the highest level cache 35c. The prediction signal is thus a hit prediction signal and is used by the instruction scheduler to schedule ahead of the instructions dependant upon the results of the load instruction additional instructions if the prediction signal indicates that a cache miss will occur.

Detailed Description Text (25):
In the event of a cache miss, the processor will retrieve data from lower level caches or main memory. While the latency for retrieval from the cache 35c may be 3 or 4 processor cycles, the latency from lower level caches can be 10 or more and from main memory can be several tens of cycles. Thus, by providing a prediction as to whether there will be a cache hit or miss, the scheduler can schedule additional instructions for cache miss predictions thereby provide useful work from the processor at a cache miss condition.

Detailed Description Text (26):
Referring now to FIG. 7, a choosing cache hit predictor 50 is shown to include a first component predictor 52, a second component predictor 56 and choosing circuit 54 as generally described for the branch predictor in conjunction with FIG. 1. The first cache hit predictor 52 operates using a first algorithm or technique and is a so-called local cache hit predictor which operates using past history of cache hits of a particular load instruction. The cache hit predictor 52 includes a cache hit past history table 53 comprised of a plurality of entries or registers appropriate for the particular implementation. Here, each of the registers are I bits wide, are right-shifting shift registers with bit zero occupying the least significant bit position and bit I-1 occupying the most significant bit position. Register file 53 is fed at an input thereof via the address from the program counter 11 (FIG. 1).

Detailed Description Text (28):
At an output of the register file 53, the contents of the selected register are used as an index or address to a second file 55. Here the second file 55 is a counter file comprised of a plurality of M-bit counters. The output of each of these counters are selectively provided in accordance with the address from the register file 53 and are fed to an input 54b of a multiplexer 54. Here, each of the counters are non-modular (i.e., non wrap-around) M-bit wide counters where M is equal to illustratively 3. The counters use their most significant bit as the output bit to the input of multiplexer 54 along line 54b. In a convention used here, the MSB provided from counter file 55 illustratively indicates that a cache hit should be predicted when the MSB equals a logic "one" or a cache miss should be predicted when the MSB equals a logic "zero".

Detailed Description Text (29):
Thus, past cache hit history table 13 is used to store the N previous cache hit/miss occurrences of that particular instruction. Whereas, the bank of non-modulo counters is used to predict whether the instruction will have a cache hit or miss. Accordingly, this technique of cache prediction examines the history of the previous N number hits/misses as provided from the past history table 53 and the contents of the associated counter 55 for a particular instruction in response to the address provided from the program counter to provide a prediction on the behavior of load-type of instructions.

Detailed Description Text (30):
The choosing predictor 50 further includes a second prediction circuit 56 which uses a technique which is selected to be different from the technique or algorithm implemented in the first cache hit predictor 52. Here, this second predictor 56 uses a global cache hit prediction technique in which the histories of the most recent load instructions encountered prior to the current load instruction are used to make a prediction concerning the current load instruction. Accordingly, the

global cache hit predictor 56 includes a global path history register 59, here 13 bits wide which stores the 13 most recent hit/miss resolutions of the 13 most previously encountered loads prior to the current load. This index stored in register 59 serves as an index to a counter file 57 which comprised of a plurality of counters. Here, each of the counters are two bit counters, are non-modular, i.e., non wrap-around, and are up/down counters. The most significant bit of a selected one of the counters which is selected in accordance with the branch path history fed to the counter file provides an input along lines 54b and 54c multiplexer 54.

Detailed Description Text (31):
The choosing predictor 50 includes a choosing circuit 60 comprised of the aforementioned multiplexer 54, as well as control logic (not shown) which is used to update or train a choosing counter table 62. In the embodiment shown in FIG. 7, the choosing counter table 62 is addressed via the program counter 11. In other embodiments, the global hits history may be used. This index serves to access the counter table 62. The counters provide their MSB as the selection bit to determine whether the multiplexer 54 will provide at the output 54a thereof a cache hit prediction from the predictor 52 or the predictor 56.

Detailed Description Text (32):
The counter 62 is updated via the logic (not shown) which has as inputs the prediction outputs from the component predictors 52, 54, as well as, a cache hit signal 66 provided from a cache controller (not shown). The logic produces an increment signal to the appropriate counter if the predictor 52 provided a correct prediction and the predictor 56 provided an incorrect prediction, and provides a decrement signal to the counter if the predictor 56 provided a correct prediction and the predictor 52 provided an incorrect prediction. If both predictors provided incorrect predictions or correct predictions, the state of a counter is not changed. Over a period of time, the training mechanism provides values for the entries in the counter table 62 which will select the appropriate one of the predictors 52 and 56 to provide optimal performance for a processor incorporating the cache hit choosing predictor 50.

CLAIMS:

3. The predictor of claim 2 wherein said means for choosing is a bank of counters responsive to the predicted instruction outcome.

15. The predictor of claim 14 wherein said means for choosing is a bank of counters responsive to a resolution of the predicted action.

Previous Doc Next Doc Go to Doc#

L21: Entry 5 of 7 File: USPT May 26, 1998

DOCUMENT-IDENTIFIER: US 5758142 A
TITLE: Trainable apparatus for predicting instruction outcomes in pipelined processors

Abstract Text (1):
A predictor which chooses between two or more predictors is described. The predictor includes a first component predictor which operates according to a first algorithm to produce a prediction of an action and a second component predictor which operates according to a second algorithm to produce a prediction of said action. The predictor also includes means, coupled to each of said first and second predictors, for choosing between predictions provided from said predictors to provide a prediction of the action from the predictor. The predictor can be used to predict outcomes of branches, cache hits, prefetched instruction sequences, and so forth.

Brief Summary Text (3):
As it is known in the art, computer systems have become ubiquitous. In particular, one type of computer system widely employed includes a so called pipelined processor. In a pipelined processor, instructions are decomposed into assembly-like stages and the instruction is operated on at each of the stages for each clock cycle in the pipeline. Illustratively, a pipelined stage includes an instruction fetch stage in which instructions are fetched in one or several cycles from a cache memory, an instruction decode stage in which an instruction's op-code, (i.e., a portion of the instruction which determines the function of the instruction) is examined to ascertain the function of the instruction and, thus, the resources needed by the instruction, as well as instruction issue stage and so forth.

Brief Summary Text (15):
Other dependencies such as data dependencies are encountered in pipelined processors. These data dependencies also require the processor to wait or flush the pipeline. One type of data dependency which occurs is a cache dependency, more commonly referred to as a cache miss.

Brief Summary Text (16):
One feature of modern processors is the use of very, high speed cache memory on or close to a semiconductor chip which embodies the central processing unit of the processor. So-called static random access memory, is used to provide a small but very fast and accessible cache for storage of data which the processor is currently or is expected to presently use. Generally, the period of time or latency from which an instruction requests data from a cache is substantially less than the time that it takes to retrieve the same data from main memory. That is, when a processor executes a load instruction, i.e., retrieves data from a cache or main memory to load a register in the processor, the processor issues an address and the address is examined by the cache. The cache generally examines a portion of the address bits against tags stored in a tag store associated with the cache to determine whether the requested data is resident in the cache. There are two basic types of caches, direct map caches and associative caches. With either type of cache, if the data requested by the processor is in the cache, a so-called cache hit is provided indicating that the cache can quickly supply the data to the processor. However, if the data is not in the cache, the data must be retrieved from either one or more

lower level (and hence slower) caches or from main memory. Thus, with a caches miss generally, the processor stalls or delays processing instructions until additional time has elapsed for the data to be retrieved from the location containing the requested data until it can be provided to the processor.

Brief Summary Text (17):
The problem associated with caches in pipelined processors employing instruction schedulers is that in the event that an instruction scheduler schedules an instruction, for example a load instruction to issue from the cache, it would be desirable to know whether or not the data will be in the cache prior to the instruction actually being executed. Therefore, a prediction technique which can predict cache hits and cache misses would be desirable. In the event a prediction technique is provided, the instruction scheduler can use this information to hold off scheduling instructions that need the load result operand, while scheduling other instructions which are indepentant of the load operand.

Drawing Description Text (8):
FIG. 6 is a block diagram of a pipelined processor employing a cache hit predictor in accordance with a still further aspect of the present invention;

Drawing Description Text (9):
FIG. 7 is a block diagram of a cache hit predictor for use in the processor shown in FIG. 6;

Detailed Description Text (10):
Therefore, the choosing predictor 10 includes a choosing circuit 20 comprised of the aforementioned multiplexer 24, as well as control logic 25, which is used to update or train a choosing table 22. In the embodiment shown in FIG. 1, the choosing table 22 is a choosing counter table and is addressed via the output of register 11a, here 13 bits of the program counter (PC) 11 for the particular decoded branch type of instruction, as previously mentioned. Alternatively, the choosing table 22 could be a register stack whose states are used to determine majority direction. In the present embodiment, the PC serves to access the counter table 22 here comprised of two bit counters 22.sub.a -22.sub.k-1. These counters provide their MSB as the selection bit to determine whether the multiplexer 24 will provide at the output 24a thereof a branch prediction from the predictor 12 or the predictor 16. The counter 22 is updated via training logic 25 which has as inputs, the prediction outputs of each component predictor 12, 16, as well as the branch resolution signal 26 provided from a branch resolution circuit not shown but which can be the type as described in the above mentioned co-pending application.

Detailed Description Text (20):
This dual stage global predictor scheme can work better than the single stage scheme 16, described in conjunction with FIGS. 1 and 2 since using history bits allows patterns of behavior to be recognized at a particular global history compared to just majority direction as is provided from just using counters.

Detailed Description Text (23):
The instructions are fed to a plurality of execution boxes (EBOX.sub.1 -EBOX.sub.i) here 34a-34i, as shown via bus 31. Each of the execution boxes EBOX.sub.1 - EBOX.sub.i includes a general purpose register file 35a arithmetic logic unit 35b and first level cache 35c and is generally of the type also described in the above-identified patent application although other types can alternatively be used. The processor 20 further includes a cache box 36 including a tag store 37a, a write buffer 37b, as well as second and third level caches 37c and 37d, and is generally of a type also described in conjunction with the above-mentioned co-pending patent application.

Detailed Description Text (24):
The processor 20 further includes a cache hit prediction choosing predictor 50, as

will be generally described in conjunction with FIG. 7. The cache hit predictor 50 is used to generate a prediction for load type of instructions as to whether data required by the load instruction is or will be available in the highest level cache 35c. The prediction signal is thus a hit prediction signal and is used by the instruction scheduler to schedule ahead of the instructions dependant upon the results of the load instruction additional instructions if the prediction signal indicates that a cache miss will occur.

Detailed Description Text (25):
In the event of a cache miss, the processor will retrieve data from lower level caches or main memory. While the latency for retrieval from the cache 35c may be 3 or 4 processor cycles, the latency from lower level caches can be 10 or more and from main memory can be several tens of cycles. Thus, by providing a prediction as to whether there will be a cache hit or miss, the scheduler can schedule additional instructions for cache miss predictions thereby provide useful work from the processor at a cache miss condition.

Detailed Description Text (26):
Referring now to FIG. 7, a choosing cache hit predictor 50 is shown to include a first component predictor 52, a second component predictor 56 and choosing circuit 54 as generally described for the branch predictor in conjunction with FIG. 1. The first cache hit predictor 52 operates using a first algorithm or technique and is a so-called local cache hit predictor which operates using past history of cache hits of a particular load instruction. The cache hit predictor 52 includes a cache hit past history table 53 comprised of a plurality of entries or registers appropriate for the particular implementation. Here, each of the registers are I bits wide, are right-shifting shift registers with bit zero occupying the least significant bit position and bit I-1 occupying the most significant bit position. Register file 53 is fed at an input thereof via the address from the program counter 11 (FIG. 1).

Detailed Description Text (28):
At an output of the register file 53, the contents of the selected register are used as an index or address to a second file 55. Here the second file 55 is a counter file comprised of a plurality of M-bit counters. The output of each of these counters are selectively provided in accordance with the address from the register file 53 and are fed to an input 54b of a multiplexer 54. Here, each of the counters are non-modular (i.e., non wrap-around) M-bit wide counters where M is equal to illustratively 3. The counters use their most significant bit as the output bit to the input of multiplexer 54 along line 54b. In a convention used here, the MSB provided from counter file 55 illustratively indicates that a cache hit should be predicted when the MSB equals a logic "one" or a cache miss should be predicted when the MSB equals a logic "zero".

Detailed Description Text (29):
Thus, past cache hit history table 13 is used to store the N previous cache hit/miss occurrences of that particular instruction. Whereas, the bank of non-modulo counters is used to predict whether the instruction will have a cache hit or miss. Accordingly, this technique of cache prediction examines the history of the previous N number hits/misses as provided from the past history table 53 and the contents of the associated counter 55 for a particular instruction in response to the address provided from the program counter to provide a prediction on the behavior of load-type of instructions.

Detailed Description Text (30):
The choosing predictor 50 further includes a second prediction circuit 56 which uses a technique which is selected to be different from the technique or algorithm implemented in the first cache hit predictor 52. Here, this second predictor 56 uses a global cache hit prediction technique in which the histories of the most recent load instructions encountered prior to the current load instruction are used to make a prediction concerning the current load instruction. Accordingly, the

global cache hit predictor 56 includes a global path history register 59, here 13 bits wide which stores the 13 most recent hit/miss resolutions of the 13 most previously encountered loads prior to the current load. This index stored in register 59 serves as an index to a counter file 57 which comprised of a plurality of counters. Here, each of the counters are two bit counters, are non-modular, i.e., non wrap-around, and are up/down counters. The most significant bit of a selected one of the counters which is selected in accordance with the branch path history fed to the counter file provides an input along lines 54b and 54c multiplexer 54.

Detailed Description Text (31):
The choosing predictor 50 includes a choosing circuit 60 comprised of the aforementioned multiplexer 54, as well as control logic (not shown) which is used to update or train a choosing counter table 62. In the embodiment shown in FIG. 7, the choosing counter table 62 is addressed via the program counter 11. In other embodiments, the global hits history may be used. This index serves to access the counter table 62. The counters provide their MSB as the selection bit to determine whether the multiplexer 54 will provide at the output 54a thereof a cache hit prediction from the predictor 52 or the predictor 56.

Detailed Description Text (32):
The counter 62 is updated via the logic (not shown) which has as inputs the prediction outputs from the component predictors 52, 54, as well as, a cache hit signal 66 provided from a cache controller (not shown). The logic produces an increment signal to the appropriate counter if the predictor 52 provided a correct prediction and the predictor 56 provided an incorrect prediction, and provides a decrement signal to the counter if the predictor 56 provided a correct prediction and the predictor 52 provided an incorrect prediction. If both predictors provided incorrect predictions or correct predictions, the state of a counter is not changed. Over a period of time, the training mechanism provides values for the entries in the counter table 62 which will select the appropriate one of the predictors 52 and 56 to provide optimal performance for a processor incorporating the cache hit choosing predictor 50.

CLAIMS:

3. The predictor of claim 2 wherein said means for choosing is a bank of counters responsive to the predicted instruction outcome.

15. The predictor of claim 14 wherein said means for choosing is a bank of counters responsive to a resolution of the predicted action.

Previous Doc Next Doc Go to Doc#